

REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Cited Art

The Action cites The Object Management Group, entitled “The Common Object Request Broker: Architecture and Specification CORBA”, Revision 2.0, July 1995 (“CORBA”).

The Action also cites Steinman, J., entitled “Incremental State Saving in Speedes Using C++”, Proceedings of the 1993 Winter Simulation Conference (“Steinman”).

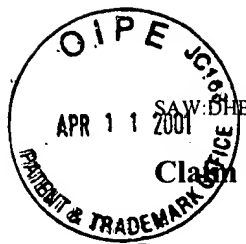
Further, the Action cites Bruce, D., “The Treatment of State in Optimistic Systems”, IEEE, pp. 40-49 (“Bruce”).

Finally, the Action cites United States Patent No. 5,889,957, to Ratner et al. (“Ratner”), entitled “Method and Apparatus for Context Sensitive Pathsend,” and United States Patent No. 5,765,174, to Bishop, entitled “System and Method for Distributed Object Resource Management” (“Bishop”).

Patentability of Claims 1-24 over CORBA, Steinman, Bruce, Ratner and Bishop, under § 103

The Office Action rejects claims 1, 3-21 under 35 U.S.C. § 103(a) as unpatentable over CORBA in view of Ratner and Steinman. The Action rejects claim 2 under 35 U.S.C. § 103(a) as unpatentable over CORBA in view of Ratner, Steinman, and Bishop. Applicants respectfully submit the claims in their present and amended form are now allowable over the cited art. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. (MPEP § 2142.)

However, the cited art does not so show. For example, with respect to claim 1, a CORBA-Ratner-Steinman combination does not teach or suggest *(1) destroying the state of the application component, (2) in response to the indication from the application component* to the operating service at the provided interface that the work is complete and *(3) without action by the client*.



Claim 1

Claim 1 is generally directed to a method of enhancing scalability of server applications and as amended recites in part,

providing an interface for the operating service to receive an indication from the application component that the work is complete;

...

destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client.

(Emphasis Added.).

For example, the Application with emphasis added states,

...As opposed to conventional object-oriented programming models where state duration is controlled solely by the client releasing its reference to the server application component instance....

Page 5, line 9-11.

...the framework provides for server application component control over state duration ...

Page 5, line 23-24.

...the component calls framework-provided interfaces...

Page 5, lines 31-32.

...The component's state thus is not left consuming resources on return from the client's call, while awaiting its final release by the client program....

Page 6, line 12-14.

...allows the server application component to have a lifetime that is independent of the client program's reference to the component...

Page 22, lines 18-20.

...the component is said to be "activated" when the component's state is in existence, and "deactivated" when the component's state is destroyed....

Page 23, 10-11.

...While deactivated, the client program 134 retains its reference to the server application component 86 indirectly through the transaction server executive 80 (i.e., the safe reference described above)....

Page 24, lines 7-10.

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

Thus, the server application component can request destruction of its own state at an interface provided by the operating service and without any action by the client. The Office Action states that paragraphs (3) and (4) of claim 1 are taught by Ratner, which states,

In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
SERVERCLASS_DIALOG_SEND_
SERVERCLASS_DIALOG_ABORT_
SERVERCLASS_DIALOG_END_

... (Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client.

... (Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.

... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client.” As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing its own destruction. Additionally, claim 1 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client....

Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established....

Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is "FeeContinue" (which causes a session to be maintained)....

Column 9, lines 58-61.

... a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ...

Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue....

Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest "destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client." The recited passage in Ratner indicates that the "message is issued by the server and received by the client." As understood by Applicant, the client (originating process) ends a "dialog" by calling "SERVERCLASS_DIALOG_END_" (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message ("FeOK") that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Accordingly, Ratner fails to teach (1) "destroying the state of the application component," (2) "in response to the indication from the application component," and (3) "without action by the client."

Next, the Office Action does not identify, and the Applicant is unable to find any description in either CORBA or Steinman that teaches the recited limitation "destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client." Further, the Office Action states that CORBA fails to teach "(3) destroying is in response to an indication and (4) without action by client." See Office Action, page 3, line 10-11.

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest "destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client."

For at least these reasons, claim 1 and its dependent claims, 3-4, should be allowed in their present form.

Dependent Claims 3 and 4

Dependent claims 3 and 4 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 3 and 4 are not belabored at this point.

Claim 2

Applicant separately addresses claim 2 in order to establish that Bishop fails to teach or suggest the following limitations,

... destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client....

(Last paragraph of amended claim 1).

... client retains a client's reference to the application component.

(Last phrase of claim 2).

The Office Action asserts that Bishop teaches only the last phrase from claim 2. However, Applicant addresses Bishop more broadly because Bishop helps establish how the prior art teaches away from the recited arrangement. Bishop in part states,

... In summary, an embodiment of the present invention is a distributed object oriented computer system in which the object oriented operating system provides for the destruction of object resources through the use of two classes of object references: strong references and weak references. Weak references (e.g., pointers) allow users to refer to an object, but does not prevent the object manager from deleting the object.

In systems using a reference counting methodology, an object is queued for deletion when the count of strong references to the object reaches a predefined threshold value. In systems using a garbage collection methodology, any object having no outstanding strong object references (with the possible exception of one reference held by the object's object manager) will be garbage collected and thereby deleted at the time of the next garbage collection cycle.... Column 1, lines 28-44.

... The present invention incurs no additional distributed communication costs to users of objects who do not wish to use the

new "weak" class of object references because those users use only the normal strong object reference mechanisms....

Column 1, lines 57-61.

... references to an object 110 stored on the file server 102 can be made from data structures within the server's main operating system domain 112, a local program 114 running in a separate domain 116 on the file server 102, as well as programs 118 running on other computer platforms in the system such as a workstation 104....

Column 3, lines 9-14.

... the MakeWeak function receives as an argument a strong object reference, which points directly or indirectly to an object. A corresponding weak object reference is generated by the MakeWeak procedure 210 by (A) ... and returning that weak object reference to the calling thread....

Column 7, lines 33-40.

... only strong object references cause an object to be retained during garbage collection....

Column 9, lines 63-64.

As understood by Applicant, Bishop describes a method whereby "users of objects" can call a procedure that returns a weak reference. Thus, a calling thread (user of an object) can reduce their holding power on an object by calling a "MakeWeak" procedure, thereby allowing the object referred to by the returned weak reference to be garbage collected.

In Bishop the client allows garbage collection by calling the make weak function. However, Bishop fails to teach or suggest (1) "destroying the state of the application component," (2) "in response to the indication from the application component ... at the provided interface," and (3) "without action by the client." In Bishop, MakeWeak is called by the client.

For at least this reason, claim 2 should be allowed in its present form.

Claim 5

Claim 5 is generally directed to scalable component-based server applications and recites in part,

the run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client.
(Emphasis Added).

For example, the Application with emphasis added states,

...the framework provides for server application component control over state duration ...

Page 5, line 23-24.

...the component calls framework-provided interfaces...

Page 5, lines 31-32.

...allows the server application component to have a lifetime that is independent of the client program's reference to the component...

Page 22, lines 18-20.

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

The Office Action refers to the references cited in claim 1 and 4, and further relies on the CORBA. However, the Office Action states in the discussion of claim 1, that CORBA does not teach “(3) destroying is in response to an indication and (4) without action by client.” See Office Action, page 3, line 10-11.

Further, as discussed above in claim 1, neither CORBA, Ratner or Steinman teaches or suggests a “run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client. For example, Ratner in part states,

...In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
SERVERCLASS_DIALOG_SEND_
SERVERCLASS_DIALOG_ABORT_
SERVERCLASS_DIALOG_END_

...

(Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client....

(Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.

... (Col. 10, lines 24-26).

As understood by Applicant, the cited passage describes an originating process creating a “dialogue” and breaking the dialogue by “either a server or client.” As understood by Applicant, the

described dialogue does not call an interface provided by an operating service thereby causing its own destruction. Additionally, claim 5 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is "FeeContinue" (which causes a session to be maintained)....

Column 9, lines 58-61.

... a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ...

Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue....

Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest a "run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client." The recited passage in Ratner indicates that the "message is issued by the server and received by the client." As understood by Applicant, the client (originating process) ends a "dialog" by calling "SERVERCLASS_DIALOG_END_" (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message ("FeOK") that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Thus, Ratner fails to teach or suggest a "run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client."

For at least these reasons, claim 5 and its dependent claims 6-12, should be allowed in their present form.

Dependent Claims 6, 7, 8, 9, 10, 11, and 12

Dependent claims 6-12 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 6-12 are not belabored at this point.

Claim 13

Claim 13 is generally directed to a method for processing work by a component with increased scalability and as amended recites in part,

receiving an indication from the component that the work by the component is complete; and

discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.

(Emphasis Added).

For example, the Application with emphasis added states,

...the framework provides for server application component control over state duration ...

Page 5, line 23-24.

...the component calls framework-provided interfaces...

Page 5, lines 31-32.

...The component's state thus is not left consuming resources on return from the client's call, while awaiting its final release by the client program....

Page 6, line 12-14.

...allows the server application component to have a lifetime that is independent of the client program's reference to the component...

Page 22, lines 18-20.

...the component is said to be "activated" when the component's state is in existence, and "deactivated" when the component's state is destroyed....

Page 23, 10-11.

...While deactivated, the client program 134 retains its reference to the server application component 86 indirectly through the transaction server executive 80 (i.e., the safe reference described above)....

Page 24, lines 7-10.

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

Thus, the server application component can request destruction of its own state at an interface provided by the operating service and without any action by the client. The Office Action

refers Applicant to the discussion of the references in claim 1 and 5. The discussion directs Applicant to Ratner, which states,

In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
SERVERCLASS_DIALOG_SEND_
SERVERCLASS_DIALOG_ABORT_
SERVERCLASS_DIALOG_END_

... (Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client.

... (Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.

... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.” As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing its own destruction. Additionally, claim 13 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client....

Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established....

Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is “FeeContinue” (which causes a session to be maintained)....

Column 9, lines 58-61.

...a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ... Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue.... Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.” The recited passage in Ratner indicates that the “message is issued by the server and received by the client.” As understood by Applicant, the client (originating process) ends a “dialog” by calling “SERVERCLASS_DIALOG_END_” (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message (“FeOK”) that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Accordingly, Ratner fails to teach or suggest “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.”

Next, the Office Action does not identify, and the Applicant is unable to find any description in either CORBA or Steinman that teaches the recited limitation “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.”

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest the recited arrangement.

For at least these reasons, claim 13 and its dependent claims 14-17, should be allowed in their present form.

Dependent Claims 14, 15, 16 and 17

Dependent claims 14-17 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 14-17 are not belabored at this point.

Claim 18

Claim 18 is generally directed to code for destroying the processing state of an application program without action from the client program and as amended recites,

code for receiving an indication from the application component that processing by the application component of the work is complete; and

code for destroying the processing state of the application program without action from the client program.
(Emphasis Added).

For example, the Application with emphasis added states,

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

The Office Action refers Applicant directly to claim 5, which for the recited passage, refers Applicant to claim 1. The Office Action states that paragraphs (3) and (4) of claim 1 are taught by Ratner which states in part,

In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
SERVERCLASS_DIALOG_SEND_
SERVERCLASS_DIALOG_ABORT_
SERVERCLASS_DIALOG_END_
...

(Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client.

... (Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.

... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “receiving an indication from the application component that processing by the application component of the work is complete” and “destroying the processing state of the application program without action from the client program.”

As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing its own destruction. Additionally, claim 18 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client....

Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established....

Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is "FeeContinue" (which causes a session to be maintained)....

Column 9, lines 58-61.

...a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ...

Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue....

Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest "receiving an indication from the application component that processing by the application component of the work is complete" and "destroying the processing state of the application program without action from the client program."

The cited passage in Ratner indicates that the "message is issued by the server and received by the client." As understood by Applicant, the client (originating process) ends a "dialog" by calling "SERVERCLASS_DIALOG_END_" (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message ("FeOK") that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Accordingly, Ratner fails to teach or suggest “receiving an indication from the application component that processing by the application component of the work is complete” and “destroying the processing state of the application program without action from the client program.”

Next, the Office Action does not identify, and the Applicant is unable to find any description in either CORBA or Steinman that teaches the recited limitation. Rather, the Office Action states that CORBA fails to teach “(3) destroying is in response to an indication and (4) without action by client.” *See* Office Action, page 3, line 10-11.

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest the recited arrangement.

For at least these reasons, claim 18 and its dependent claims 19-20, should be allowed in their present form.

Dependent Claims 19 and 20

Dependent claims 19 and 20 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 19 and 20 are not belabored at this point.

Claim 21

Claim 21 is generally directed to a method for enhancing the scalability of the server applications and recites in part,

destroying the state by the operating service in response to an indication from the application component without action by the client, such that the destroyed state is not persistent.

For example, the Application with emphasis added states,

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

The Office Action states that Ratner teaches “(2) destroying is in response to an indication from application component.” Applicant is directed to Ratner which states,

... Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “destroying the state by the operating service in response to an indication from the application component without action by the client.” As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing the destruction of the state of the application component without action by the client. Additionally, claim 18 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client....

Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established....

Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is “FeContinue” (which causes a session to be maintained)....

Column 9, lines 58-61.

...a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ...

Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue....

Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest “destroying the state by the operating service in response to an indication from the application component without action by the client.”

The cited passage in Ratner indicates that the “message is issued by the server and received by the client.” As understood by Applicant, the client (originating process) ends a “dialog” by calling “SERVERCLASS_DIALOG_END_” (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message (“FeOK”) that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Accordingly, Ratner fails to teach or suggest “destroying the state by the operating service in response to an indication from the application component without action by the client.”

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest the recited arrangement.

For at least these reasons, claim 21 should be allowed in its present form.

Claim 22

Claim 22 is generally directed to a method of enhancing scalability of server applications and recites in part,

providing an interface for the operating service to receive an indication from the application component that the work is complete;
in response to the indication and without client action,
destroying the component data state by the operating service upon
a next return of the application component from a method
invocation of the client, while persistently maintaining the work
data state.
(Emphasis Added).

For example, the Application with emphasis added states,

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

The Office Action resolves the emphasized portions of claim 22 to the discussion of claims 1 and 4. For example, Applicant is directed to Ratner which states,

In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
 SERVERCLASS_DIALOG_SEND_
 SERVERCLASS_DIALOG_ABORT_
 SERVERCLASS_DIALOG_END_

... (Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client.

... (Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “providing an interface for the operating service to receive an indication from the application component that the work is complete” and “in response to the indication and without client action, destroying the component data state by the operating service.” As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing the destruction of its own state. Additionally, claim 18 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client....

Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established....

Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is “FeContinue” (which causes a session to be maintained)....

Column 9, lines 58-61.

...a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ...

Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is

deallocated and the LACB is marked available. This effectively
kills the context sensitive dialogue.... Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest “providing an interface for the operating service to receive an indication from the application component that the work is complete” and “in response to the indication and without client action, destroying the component data state by the operating service.” The cited passage in Ratner indicates that the “message is issued by the server and received by the client.” As understood by Applicant, the client (originating process) ends a “dialog” by calling “SERVERCLASS_DIALOG_END_” (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message ("FeOK") that is received by the client (Col. 10, lines 17-23). In either case, the client takes action.

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest the recited arrangement.

For at least these reasons, claim 22 and its dependent claim 23, should be allowed in their present form.

Dependent Claim 23

Dependent claim 23 recites additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claim 23 is not belabored at this point.

Claim 24

Claim 24 is generally directed to a method of enhancing scalability of server applications and recites in part,

... providing an interface for the operating service to receive an indication from the application component that the work is complete;
... in response to the indication and without client action,
destroying the component data state by the operating service upon
a next return of the application component from a method
invocation of the client, while persistently maintaining the work
data state.
(Emphasis Added).

For example, the Application with emphasis added states,

...the server application component 86 may request deactivation before returning from processing the client program's call...

Page 33, lines 4-5.

...On return from the client's method call, the transaction server executive 80 deactivates the component, causing its state to be destroyed....

Page 36, lines 9-11.

The Office Action asserts that the emphasized portions of claim 24 are covered by the discussion of claims 1. For example, Applicant is directed to Ratner which states,

In a preferred embodiment the following Application Programmatic Interfaces (APIs) have been invented to allow for the originating process to create a dialogue, to accomplish I/O on the dialogue, to carry transactional context on I/O within the dialogue, and to terminate the dialogue:

SERVERCLASS_DIALOG_BEGIN_
SERVERCLASS_DIALOG_SEND_
SERVERCLASS_DIALOG_ABORT_
SERVERCLASS_DIALOG_END_

... (Col. 6, line 66 – Col. 7, line 8).

A dialogue (session) can be broken by either a server or a client.

... (Col. 4, lines 36-37).

Finally, a DIALOG END will force Linkmon and the operating system to clean up system resources. A DIALOG END can follow a FeOK message.... (Col. 10, lines 24-26).

As understood by Applicant, Ratner fails to teach or suggest “providing an interface for the operating service to receive an indication from the application component that the work is complete” and “in response to the indication and without client action, destroying the component data state by the operating service.” As understood by Applicant, the described dialogue does not call an interface provided by an operating service thereby causing the destruction of its own state. Additionally, claim 24 does not recite the creation or destruction of a communications session (dialogue).

Further, Ratner describes termination of a communications session by a server, as follows:

... the ability of the runtime to bind the originating process to the receiving process for the duration of the dialogue

Column 2, lines 6-7.

...The originating process issues the SERVERCLASS_DIALOG_BEGIN_ procedure call, targeting a specific type of receiving process. The run-time environment will begin a receiving process and create a logical connection between the two cooperating processes (the originating process and the receiving process)....

Column 7, lines 14-20.

...A dialogue (session) can be broken by either a server or a client.... Column 4, lines 36-37.

... a sort of temporary dedicated communication between client and server is established.... Column 9, lines 52-54.

... the first part of the protocol is the response by a server to a Dialogue Begin or Send. Most of the time the server response is "FeContinue" (which causes a session to be maintained)....

Column 9, lines 58-61.

...a return code of FeContinue will cause the run-time environment to maintain the connection between the originating process and the receiving process ... Column 10, lines 1-4.

... However, if and when an FeOK error code message is issued by the server and received by the client (via Linkmon), not only are linkages (or bindings) 603 and 605 broken, but also linkages 607 are broken, and furthermore the Dialog Control Block is deallocated and the LACB is marked available. This effectively kills the context sensitive dialogue.... Column 10, lines 17-23.

As understood by Applicant, Ratner fails to teach or suggest "providing an interface for the operating service to receive an indication from the application component that the work is complete" and "in response to the indication and without client action, destroying the component data state by the operating service." The cited passage in Ratner indicates that the "message is issued by the server and received by the client." As understood by Applicant, the client (originating process) ends a "dialog" by calling "SERVERCLASS_DIALOG_END_" (Col. 7, lines 1-8), and the server ends a dialogue by sending a return message ("FeOK") that is received by the client (Col. 10, lines 17-23). In either case, the client takes action. Accordingly, Ratner fails to teach or suggest the recited arrangement.

Thus, as understood by Applicant, a Ratner-CORBA-Steinman combination fails to teach or suggest the recited arrangement.

For at least these reasons, claim 24 should be allowed in its present form.

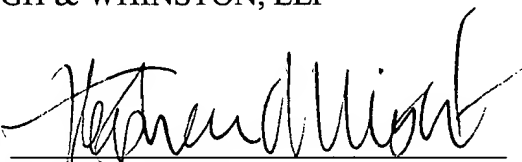
CONCLUSION

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By



Stephen A. Wight
Registration No. 37,759

RECEIVED
APR 17 2001
Group 2100

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

cc: Client (80683.1USORG)



**Marked-up Version of Amended Claims
Pursuant to 37 C.F.R. §§ 1.121(b)-(c)**

PATENT
Atty. Ref. No. 3382-47280

RECEIVED
APR 17 2001
Group 8100

1. (Thrice Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

- executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
- providing an interface for the operating service to receive an indication from the application component that the work is complete;
- maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and
- destroying the state of the application component [by the operating service] in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client.

13. (Once Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:

- encapsulating function code and a processing state for the work in a component;
- providing a reference through an operating service for a client [program] to call the function code of the component to initiate processing of the work by the component;
- receiving an indication from the component that the work by the component is complete;
- and
- discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.

14. (Once Amended) The method of claim 13 further comprising:

- performing the step of discarding the processing state upon a next return of the component from a call of the client [program] following the indication from the component that the work is complete.

21. (Twice Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;

maintaining the state in the main memory between the method invocations of the function code by the client in the absence of an indication from the application component that the work is complete; and

destroying the state by the operating service in response to an indication from the application component without action by the client, such that the destroyed state is not persistent.

22. (Once Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

executing an application component under control of an operating service, the application component having a component data state and function code for performing work on a data resource responsive to method invocations from a client, the component's work on the data resource having a work data state, the component data state initially having an initial post-creation state upon the component's creation;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the component data state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

in response to the indication and without client action, destroying the component data state by the operating service [system] upon a next return of the application component from a method invocation of the client, while persistently maintaining the work data state.

24. (Once Amended) A computer readable program code-carrying media having software program code encoded thereon, the software program code for executing on a server computer and implementing a method of enhancing scalability of server applications, the method comprising:

executing an application component under control of an operating service, the application component having a component data state and function code for performing work on a data resource responsive to method invocations from a client, the component's work on the data resource having a work data state, the component data state initially having an initial post-creation state upon the component's creation;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the component data state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

in response to the indication and without client action, destroying the component data state by the operating service [system] upon a next return of the application component from a method invocation of the client, while persistently maintaining the work data state.